Technical Informatics I

Control Structures (Repetition)
`for` loops

by
Dr. Fatimah
Faculty of Mechanical Engineering
fatimahd@ump.edu.my

Control Structures (Repetition)

- Aims
  - Introduce students to Control Structures (Repetition): `for`
  - Introduce students to increment/decrement operators
  - Introduce students to jump statements: `break` and `continue` statements

- Expected Outcomes
  - Students are able to construct simple C programs that can implement repetition control structures `for` while incorporating increment/decrement operators and jump statements where necessary

- References
  - Harry H. Cheng, 2010. C for Engineers and Scientists: An Interpretive Approach, McGraw Hill

Content

- Selection Structures: `for`
- Increment/Decrement Operators
- Jump statements
  - Break Statements
  - Continue Statements
- Nested Loops
  - Nested `for` loops
- Examples
- Conclusion

Control structures

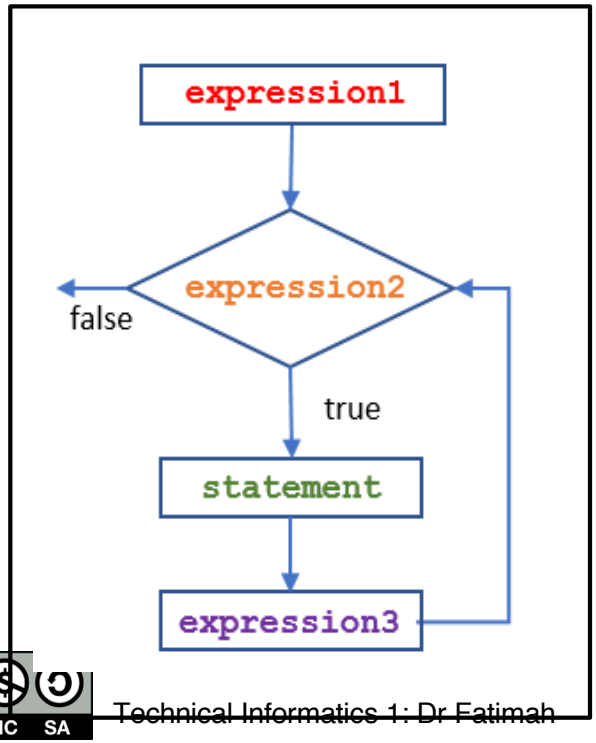
- There are 3 control structures for C programs:
  1. **Sequence**
     - Each statement is executed sequentially (as seen in the previous lectures
  2. **Selection**
     - One statement is *selected* over another depending on a Selection
       - If, else if, else & switch
       - If var1 > 10, do *this*…, else do *that*…
  3. **Repetition**
     - Statements are *repeatedly* executed until it meets a certain *condition*
       - for, while, do-while loops

Control Structure – Repetition (Overview)

- Three kinds of selections structures
  - **`while`** Loop
  - **`do-while`** Loop
  - **`for`** Loop
- Loops Consist of
  - 1: Loop Initialization
  - 2: Terminating Condition (Loop Guard)
  - 3: Loop Body (Statement)
  - 4: Terminating Action

Control Structure (Repetition): `for`

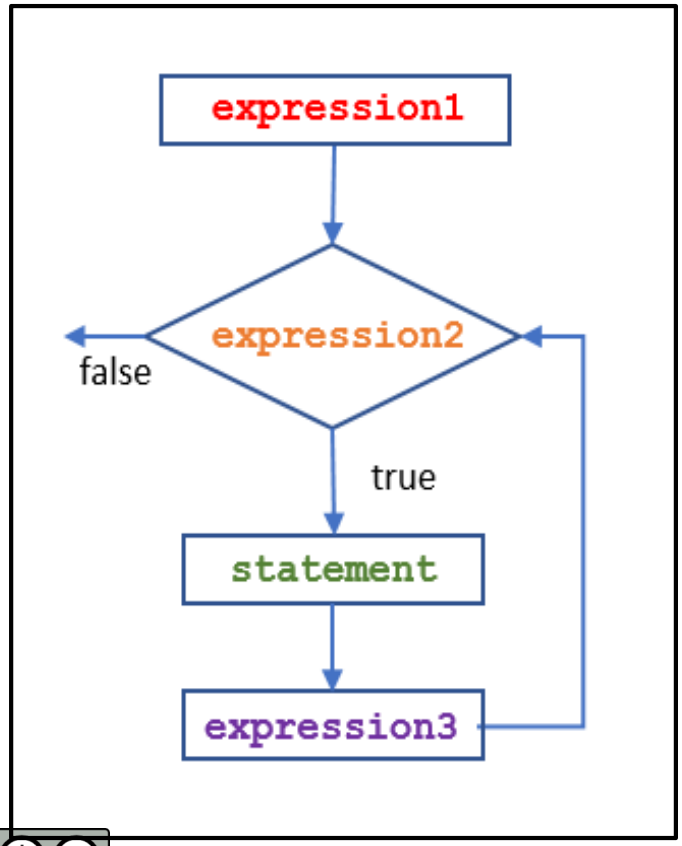
- The flow chart and the syntax of a `for` loop is given as follows where **`statement`** will be executed until **`expression`** is FALSE.




```
for (expression1; expression2; expression3) {
    statement
}
```

Control Structure (Repetition): `for`

- The syntax of a **`for`** statement is as follows:




```
for (expression1; expression2; expression3) {
    statement
}
```

Loop Initialization

Loop Guard

Loop initialization

Control Structure (Repetition): `for`

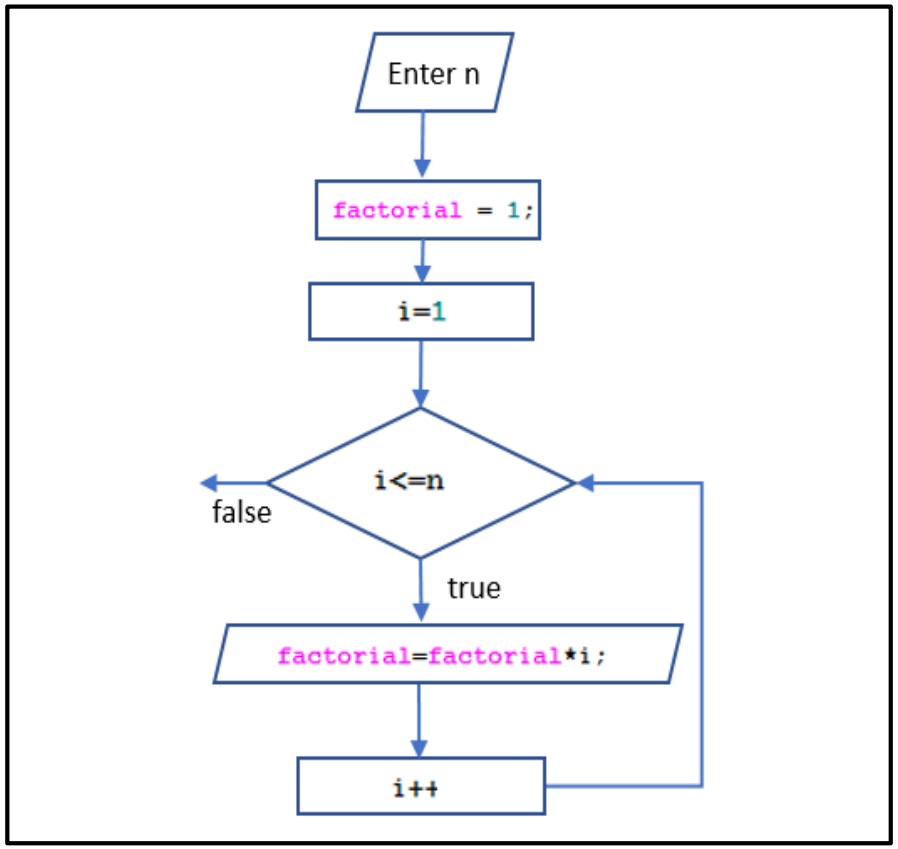
- **Example 1:**

Write a code that calculates the factorial of n where n is the user input using a `for` loop.

For example: If the use inputs n = 5, then the code will return 120 since

$$5! = 1*2*3*4*5 = 120$$

Control Structure (Repetition): `for`




```
scanf("%d",&n);
factorial = 1;
for(i=1; i<=n ; i++){
    factorial=factorial*i;
}
```

Control Structure (Repetition): `for`

```c
#include <stdio.h>

int main(){
    int i, n, factorial;

    printf("Enter n:\n");
    scanf("%d",&n); /*take in user input*/

    factorial = 1; /*initialize var factorial*/

    for(i=1; i<=n ; i++){
        factorial=factorial*i; /*statement to calc factorial*/
    }

    printf("Factorial: %d! = %d\n", n, factorial);

    return 0;
}
```

```
Enter n:
3
The result of 3! = 6
>Exit code: 0
```

```
Enter n:
4
The result of 4! = 24
>Exit code: 0
```

```
Enter n:
12
The result of 12! = 479001600
>Exit code: 0
```

Increment and Decrement Operators

- `for` loops commonly use
  - increment (**++**): increases the operand value
  - decrement (**--**): decreases the operand value
- For example:

```
for(i=1;i<=n;i++){
  factorial = factorial*i;
}
```

The variable `i` increase one after each iteration

Increment and Decrement Operators

- **Example 2:**

```
1   /* Example 4a: For Loops */
2   /*Prints out 0 to 4 using for loops*/
3   #include <stdio.h>
4   int main() {
5       int i;
6
7       for(i=0; i<5; i++) {
8       printf("%d  ", i);
9       }
10
11      return 0;
12  }
```

Output:

```
>ch -u "L7-example4a.c"
0  1  2  3  4  >Exit code: 0
```

Jump Statements

- **`break` Statements**
  - Allows us to 'break' out from the loop for an early exit (before meeting the loop guard condition)
- **`continue` statements**
  - The `continue` statement skips the rest of the statements in its current iteration and jumps to the next iteration of the enclosing loop
  - A continue statement should only appear in a loop body.

Jump Statements

- **Break Statements**
  - Allows us to 'break' out from the loop for an early exit (before meeting the loop guard condition)

- **Example 2a: Break Statements**

```
1   /* Example 4b: Break Statement */
2   #include <stdio.h>
3   int main() {
4       int i;
5
6       for(i=0; i<5; i++) {
7         if(i == 3) {
8           break;
9         }
10      printf("%d  ", i);
11      }
12
13      return 0;
14  }
```

Output:

```
>ch -u "L7-example4b.c"
0  1  2  >Exit code: 0
```

Jump Statements

- **Continue Statements**
  - The `continue` statement skips the rest of the statements in its current iteration and jumps to the next iteration of the enclosing loop
- **Example 2b: Continue Statements**

```
1   /* Example 4c: Continue Statements */
2   #include <stdio.h>
3   int main() {
4       int i;
5
6       for(i=0; i<5; i++) {
7         if(i == 3) {
8           continue;
9         }
10      printf("%d  ", i);
11      }
12
13      return 0;
14  }
```

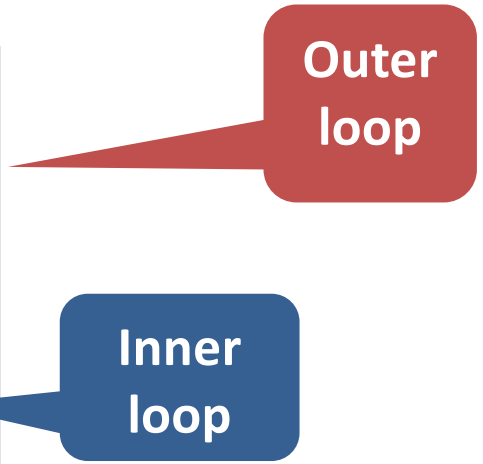
Output:

```
>ch -u "L7-example4c.c"
0  1  2  4  >Exit code: 0
```

Nested Loops

- Nested loops consist of an **outer loop** with one or more **inner loops**.
- e.g. for loop:

```
for (i=1;i<=3;i++){
    for(j=1;j<=4;j++){
        …
    }
}
```



- The above loop will run for 3*4 iterations.

Nested Loops

- **Example 3:**

Write a program to print a multiplication table:

```
>ch -u "L7-example5.c"
        1   2   3   4   5   6   7   8   9  10
   ----------------------------------------
   1|   1
   2|   2   4
   3|   3   6   9
   4|   4   8  12  16
   5|   5  10  15  20  25
   6|   6  12  18  24  30  36
   7|   7  14  21  28  35  42  49
   8|   8  16  24  32  40  48  56  64
   9|   9  18  27  36  45  54  63  72  81
  10|  10  20  30  40  50  60  70  80  90 100
   ----------------------------------------
>Exit code: 0
```


Example adapted from Cheng, 2010

Nested Loops

- **Example 3:**

```
/* Prints out multiplication table*/

#include <stdio.h>

int main() {
    int i, j;

    printf("        1   2   3   4   5   6   7   8   9  10\n");
    printf("   ----------------------------------------\n");
    for(i=1; i<= 10; i++) {    /* outer loop: prints out 1-10 in first column) */
        printf("%4d|", i);
        for(j=1; j<=i; j++) {   /* inner loop: prints out values of multiplication row by row */
            printf("%4d", i*j);
        }
        printf("\n");
    }
    printf("   ----------------------------------------\n");
    return 0;
}
```

Nested Loops

- **Example 4:**

Calculate the function

$$f_2(x,y) = \frac{\sin(\sqrt{x^2+y^2})}{\sqrt{x^2+y^2}}$$

for x from -10 <= x <= 10 with a step size 10 and

y from -10 <= y <= 10 with a step size of 10.

Nested Loops

- **Example 5:**

```c
/* Example 6: Nested Loops */
#include <stdio.h>
#include <math.h>

int main() {
    double x, x0=-10.0, xf=10.0, xstep=10.0,
           y, y0=-10.0, yf=10.0, ystep=10.0, result;
    int i, j, nx, ny;

    printf("      x          y      f2(x,y)\n");
    printf("  ----------------------------\n");

    nx  =  (xf - x0)/xstep + 1; /* num of points for x */
    ny  =  (yf - y0)/ystep + 1; /* num of points for y */

    for(i = 0; i < nx; i++) {
       x = x0 + i*xstep;          /* calculate value for x */
       for(j = 0; j <ny; j++) {
          y = y0 + j*ystep;       /* calculate value for y */
          result = sin(sqrt(x*x + y*y))/sqrt(x*x + y*y);
          printf("%10.4f %10.4f %8.4f\n", x, y, result);
       }
    }
    return 0;
}
```

Output:

```
>ch -u "L7-example6.c"
      x          y      f2(x,y)
  ----------------------------
  -10.0000   -10.0000   0.0707
  -10.0000     0.0000  -0.0544
  -10.0000    10.0000   0.0707
    0.0000   -10.0000  -0.0544
    0.0000     0.0000 nan
    0.0000    10.0000  -0.0544
   10.0000   -10.0000   0.0707
   10.0000     0.0000  -0.0544
   10.0000    10.0000   0.0707
>Exit code: 0
```

Note that sin(0)/0 is NaN

Nested Loops

Note:

1. The header file **`math.h`** is required to use the function `sin(x)`
2. Loop control variable **MUST** be of type `int`.
3. The format specifier "`%8.4`" with implies field width 8 and 4 digits after the decimal point.

4. In order to generate data points in the range `x0 <= x <= xf` with linear step size, `xstep`, the number of points is given by:

```
n = (xf - x0)/xstep + 1;
```

Where each data point can be calculated by:

```
x = x0 + i*xstep;
```

Technical Informatics I

Lecture 7

Dr Fatimah